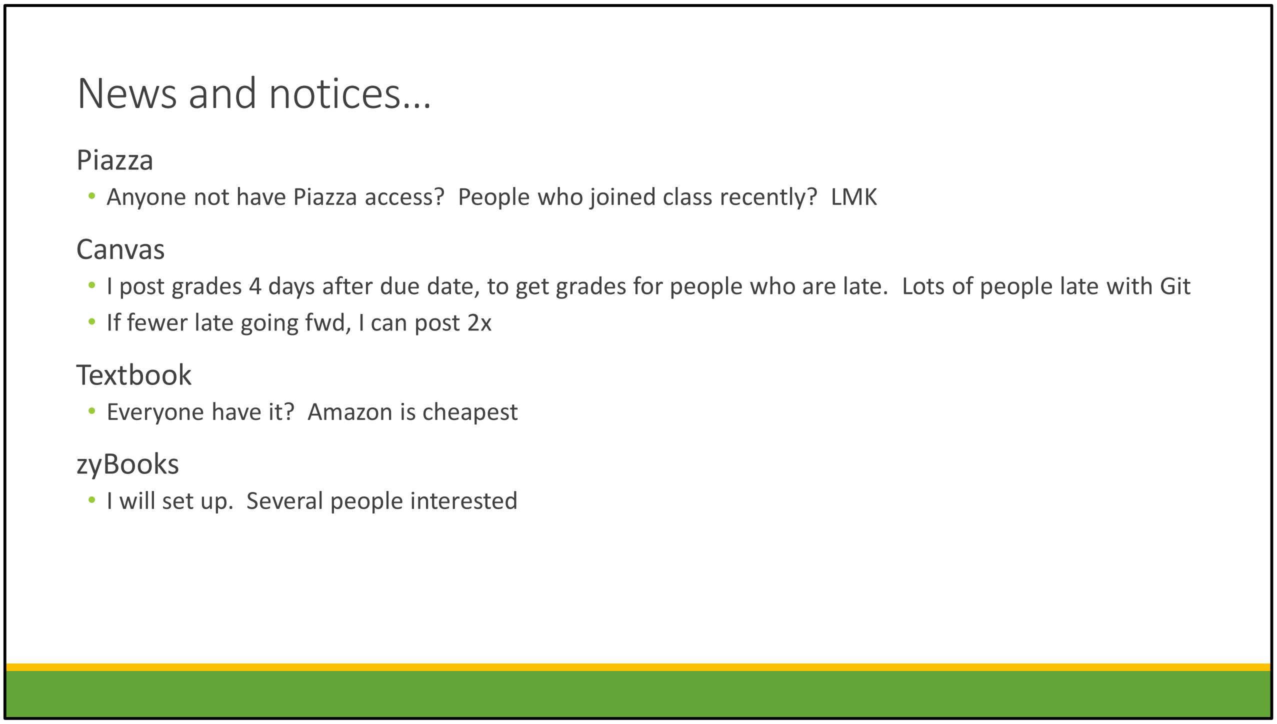
CS112 – Java Programming

Fall 2022



News and notices…

Piazza

- Anyone not have Piazza access? People who joined class recently? LMK

Canvas

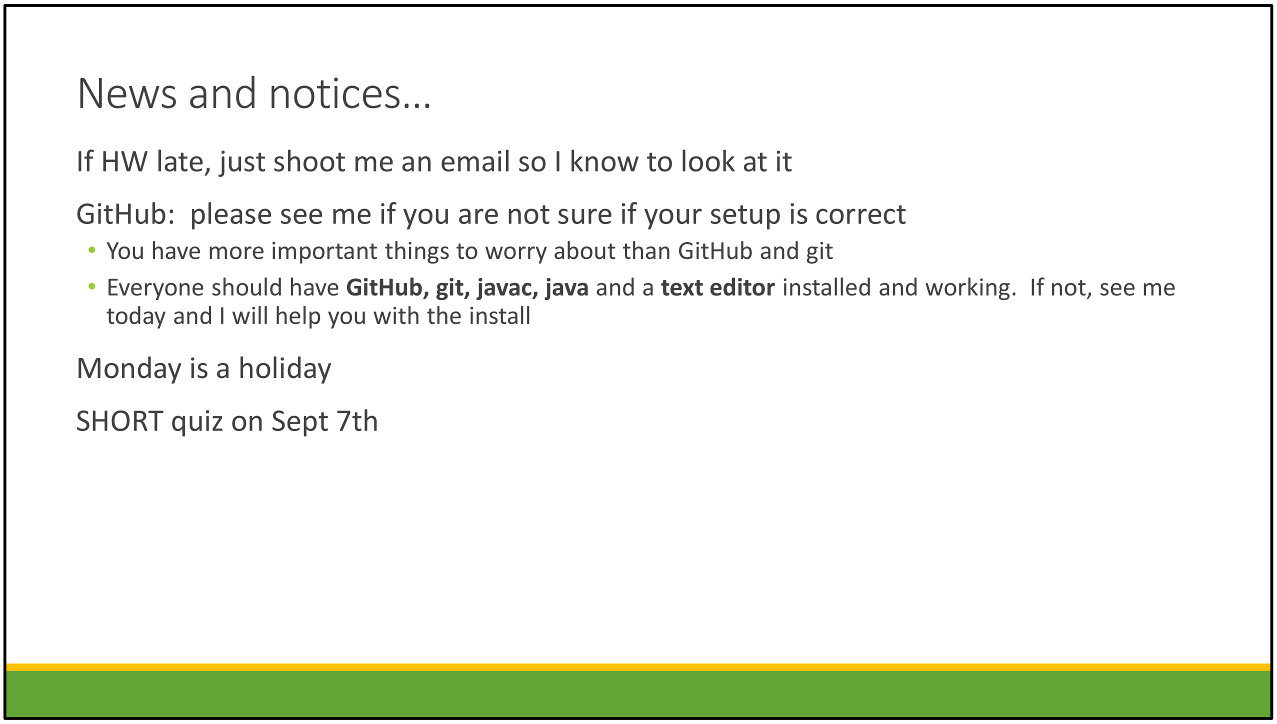
- I post grades 4 days after due date, to get grades for people who are late. Lots of people late with Git
- If fewer late going fwd, I can post 2x

Textbook

- Everyone have it? Amazon is cheapest

zyBooks

- I will set up. Several people interested



News and notices…

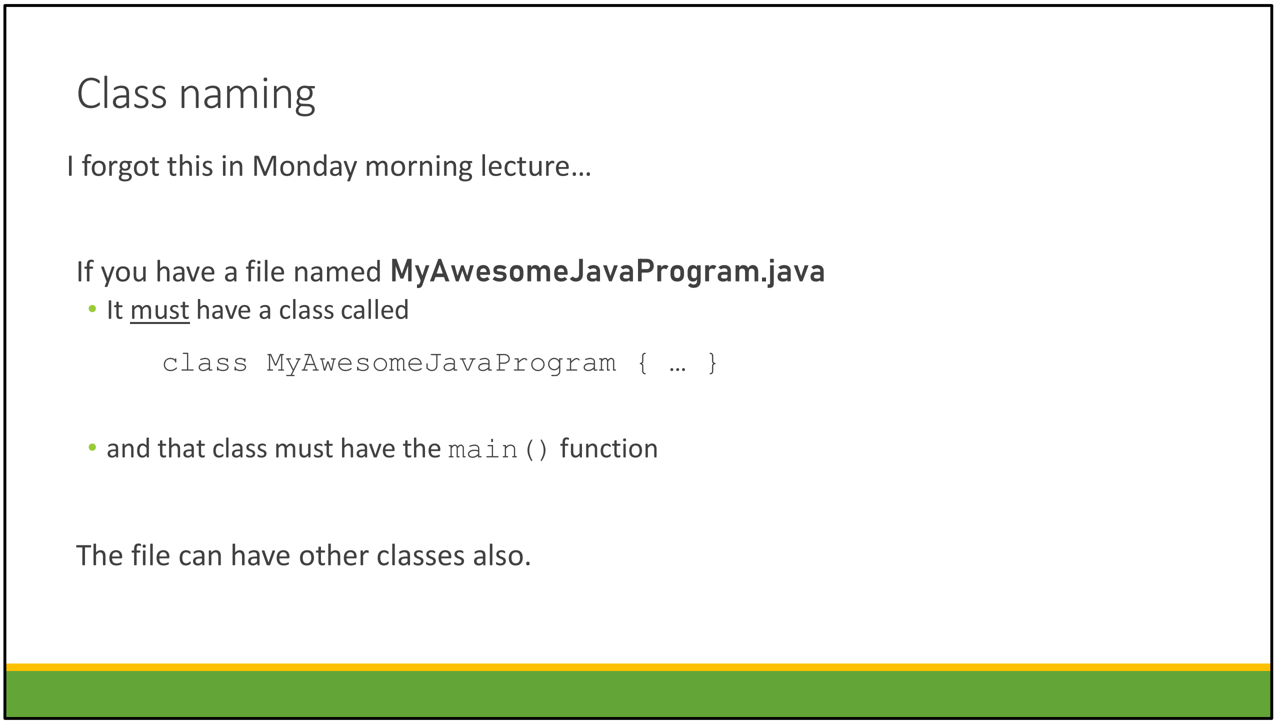
If HW late, just shoot me an email so I know to look at it

GitHub: please see me if you are not sure if your setup is correct

- You have more important things to worry about than GitHub and git
- Everyone should have **GitHub, git, javac, java** and a **text editor** installed and working. If not, see me today and I will help you with the install

Monday is a holiday

SHORT quiz on Sept 7th



Class naming

I forgot this in Monday morning lecture…

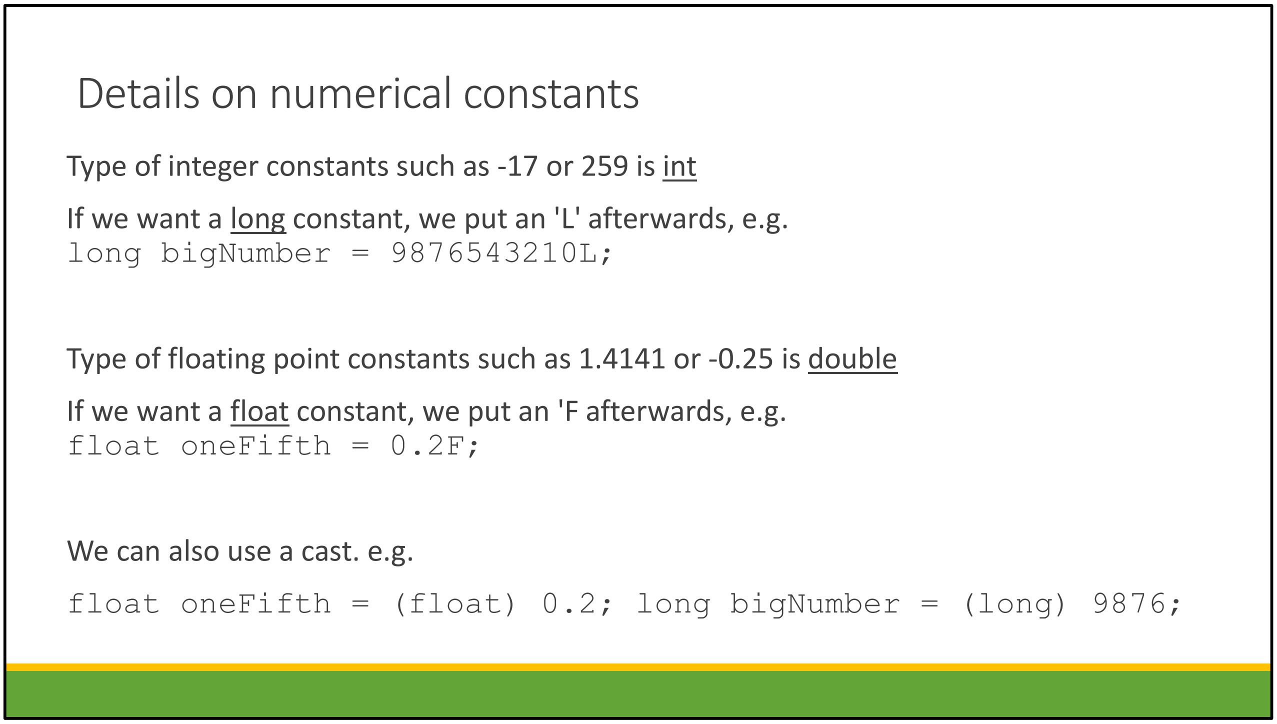
If you have a file named **MyAwesomeJavaProgram.java**

- It must have a class called

```
class MyAwesomeJavaProgram { … }
```

- and that class must have the `main()` function

The file can have other classes also.



Details on numerical constants

Type of integer constants such as -17 or 259 is int

If we want a long constant, we put an 'L' afterwards, e.g.

```
long bigNumber = 9876543210L;
```

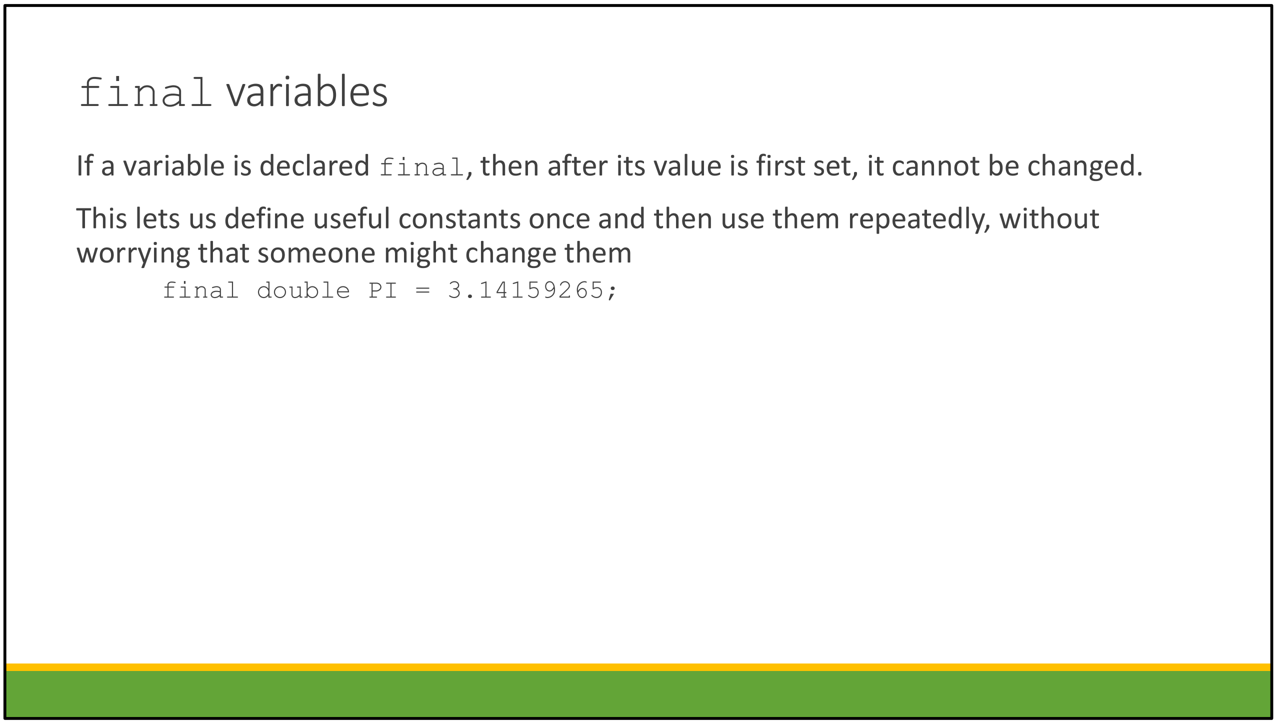
Type of floating point constants such as 1.4141 or -0.25 is double

If we want a float constant, we put an 'F afterwards, e.g.

```
float oneFifth = 0.2F;
```

We can also use a cast. e.g.

```
float oneFifth = (float) 0.2; long bigNumber = (long) 9876;
```



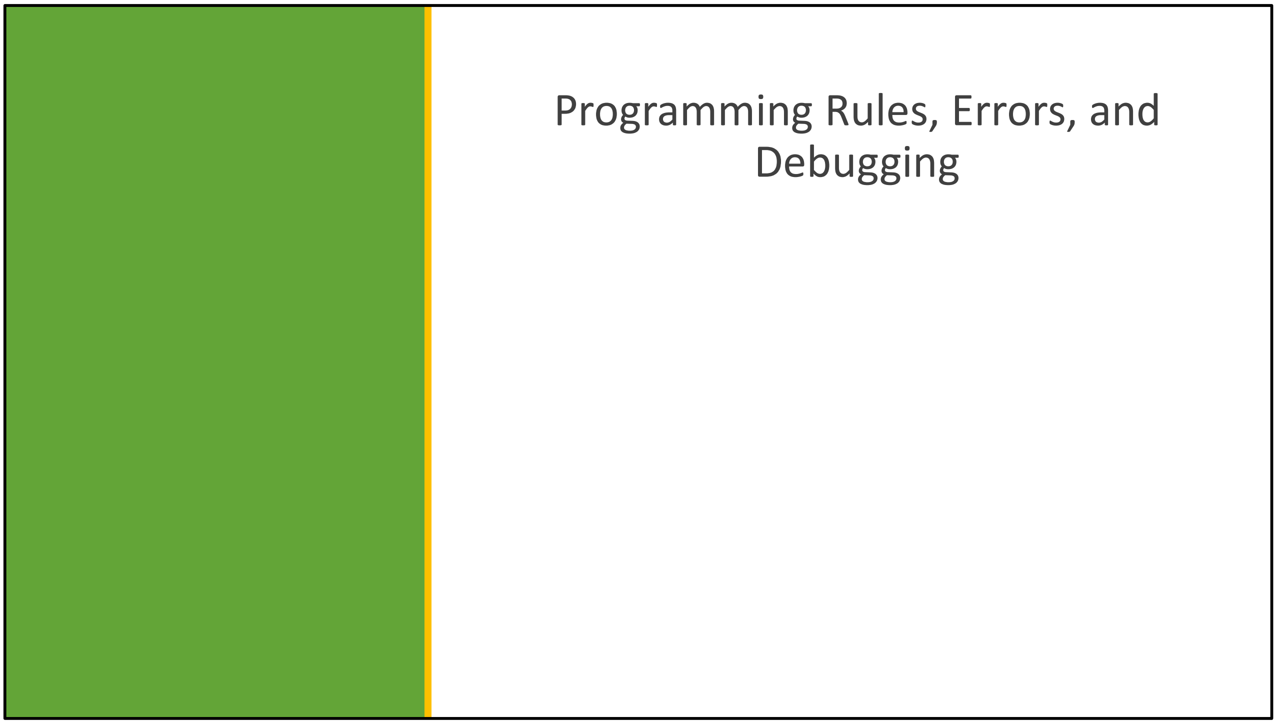
`final` variables

If a variable is declared `final`, then after its value is first set, it cannot be changed.

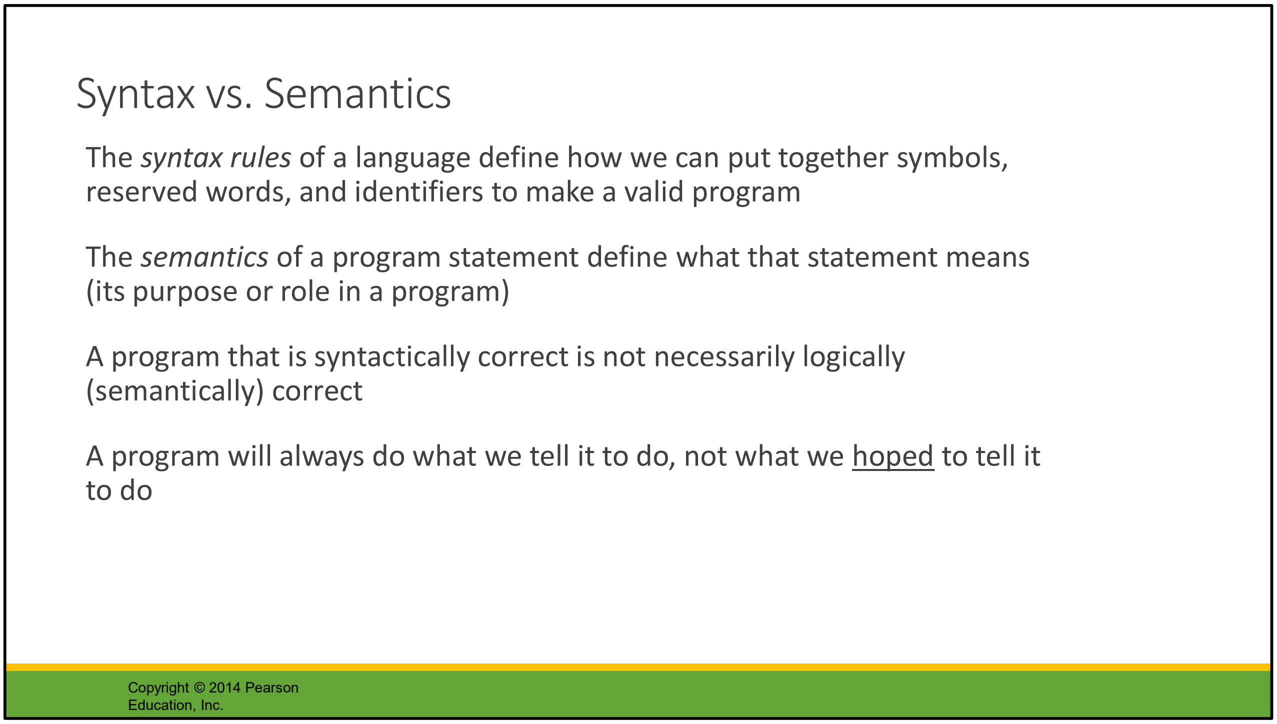
This lets us define useful constants once and then use them repeatedly, without worrying that someone might change them

```
      final double PI = 3.14159265;
```

Someone like 'ourselves' by accident



Programming Rules, Errors, and Debugging



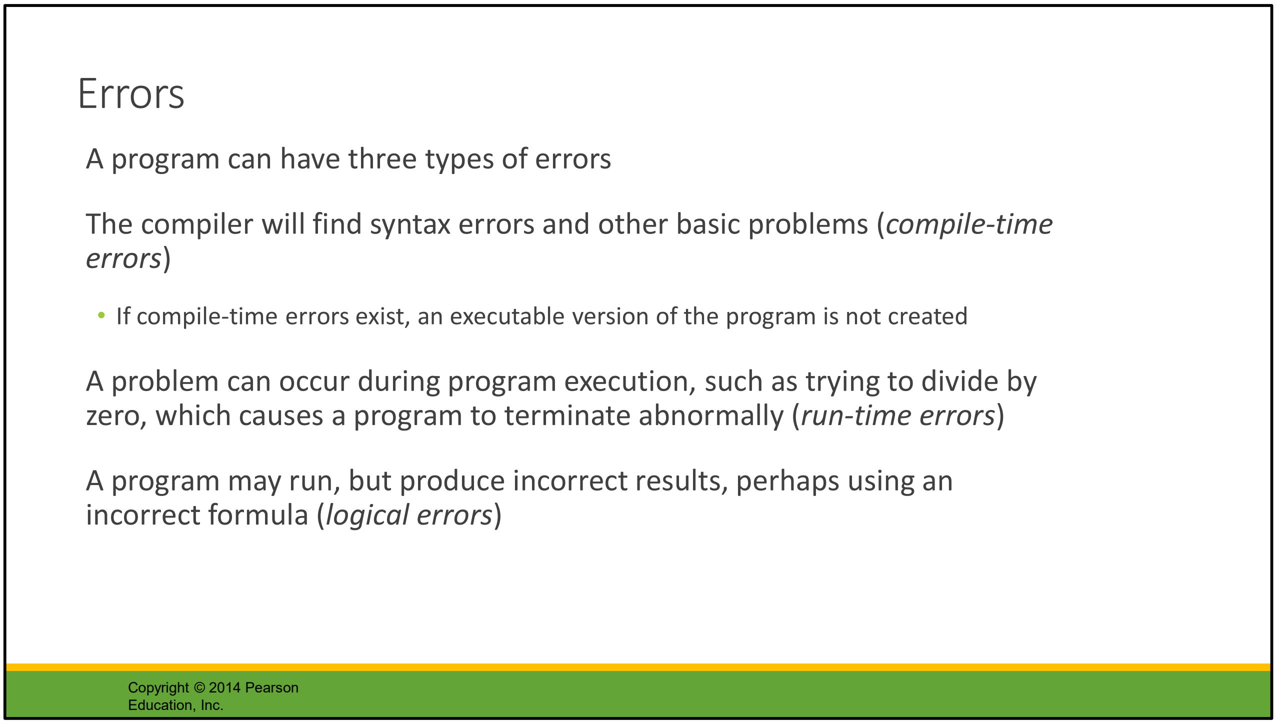
Syntax vs. Semantics

The *syntax rules* of a language define how we can put together symbols, reserved words, and identifiers to make a valid program

The *semantics* of a program statement define what that statement means (its purpose or role in a program)

A program that is syntactically correct is not necessarily logically (semantically) correct

A program will always do what we tell it to do, not what we hoped to tell it to do



Errors

A program can have three types of errors

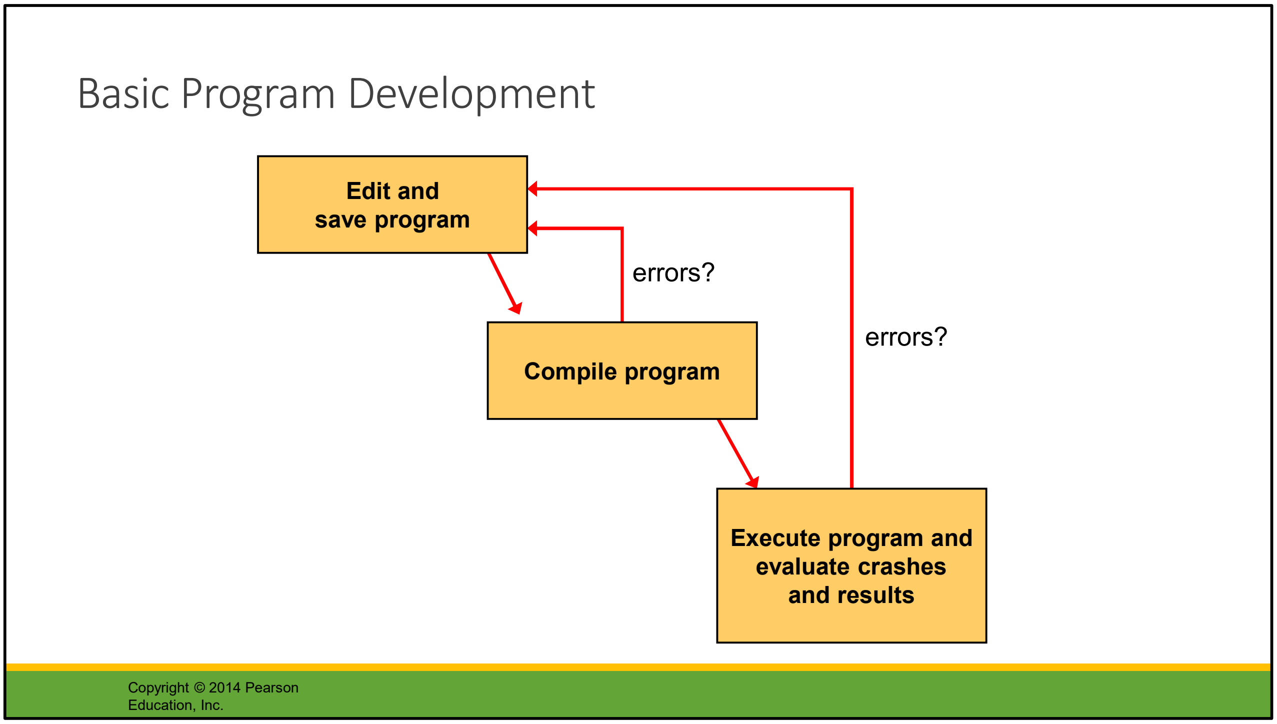
The compiler will find syntax errors and other basic problems (*compile-time errors*)

- If compile-time errors exist, an executable version of the program is not created

A problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally (*run-time errors*)

A program may run, but produce incorrect results, perhaps using an incorrect formula (*logical errors*)

"crash" = terminate abnormally



Basic Program Development

Edit and save program

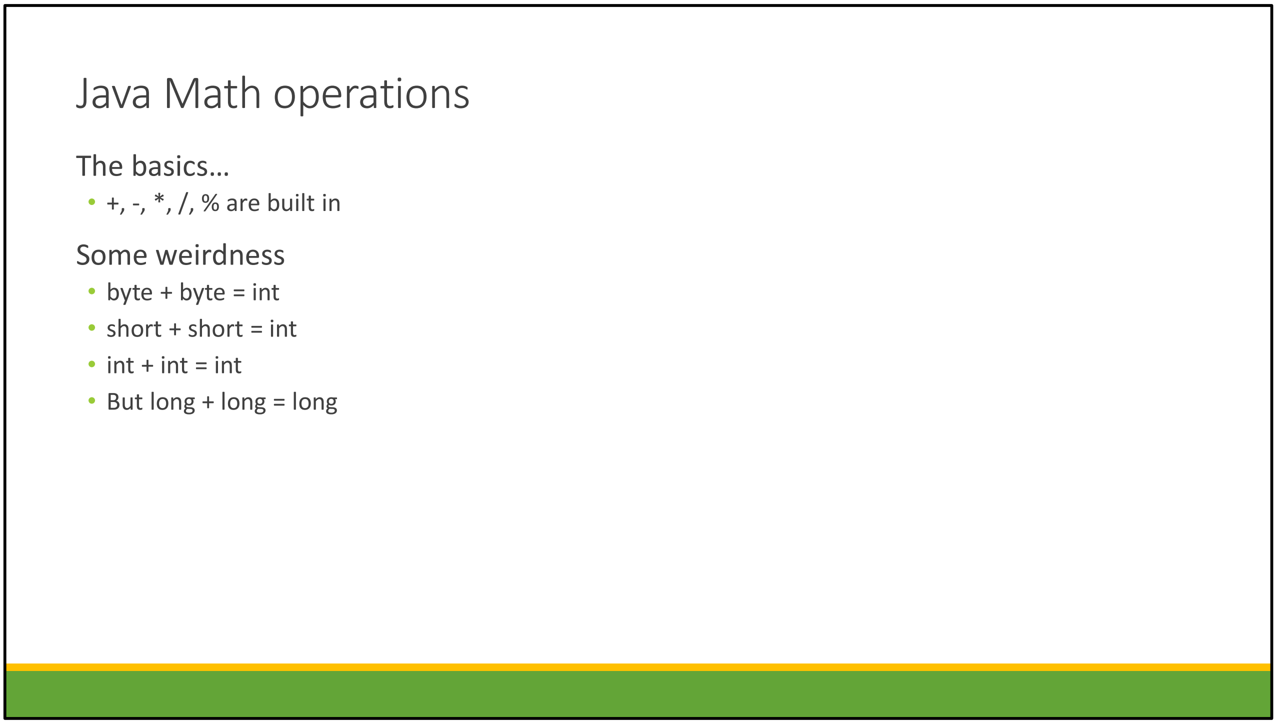
Compile program

errors?

Execute program and evaluate crashes and results

errors?

Awfully nice when compiler finds errors for us.



Java Math operations

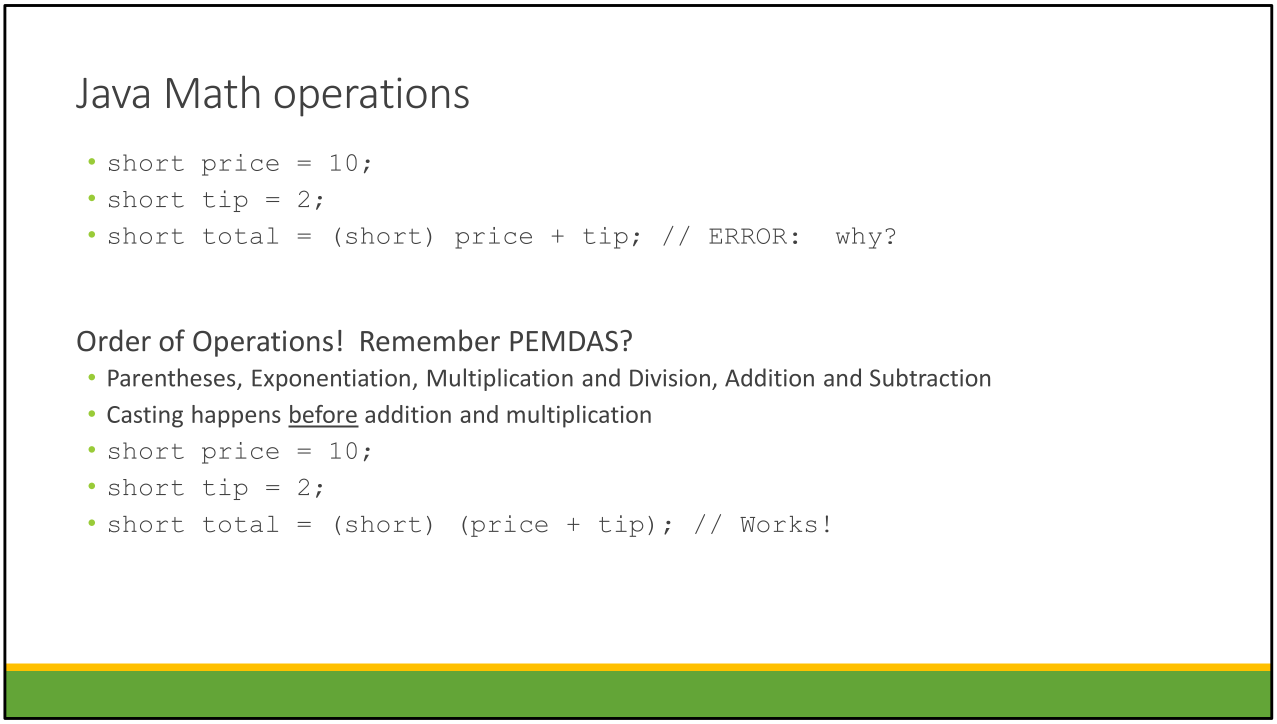
The basics…

- +, -, *, /, % are built in

Some weirdness

- byte + byte = int
- short + short = int
- int + int = int
- But long + long = long

% is remainder. Only works with integer data types.



Java Math operations

- `short price = 10;`
- `short tip = 2;`
- `short total = (short) price + tip; // ERROR:  why?`

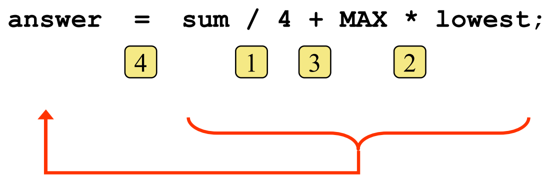
Order of Operations! Remember PEMDAS?

- Parentheses, Exponentiation, Multiplication and Division, Addition and Subtraction
- Casting happens before addition and multiplication
- `short price = 10;`
- `short tip = 2;`
- `short total = (short) (price + tip); // Works!`

Assignment

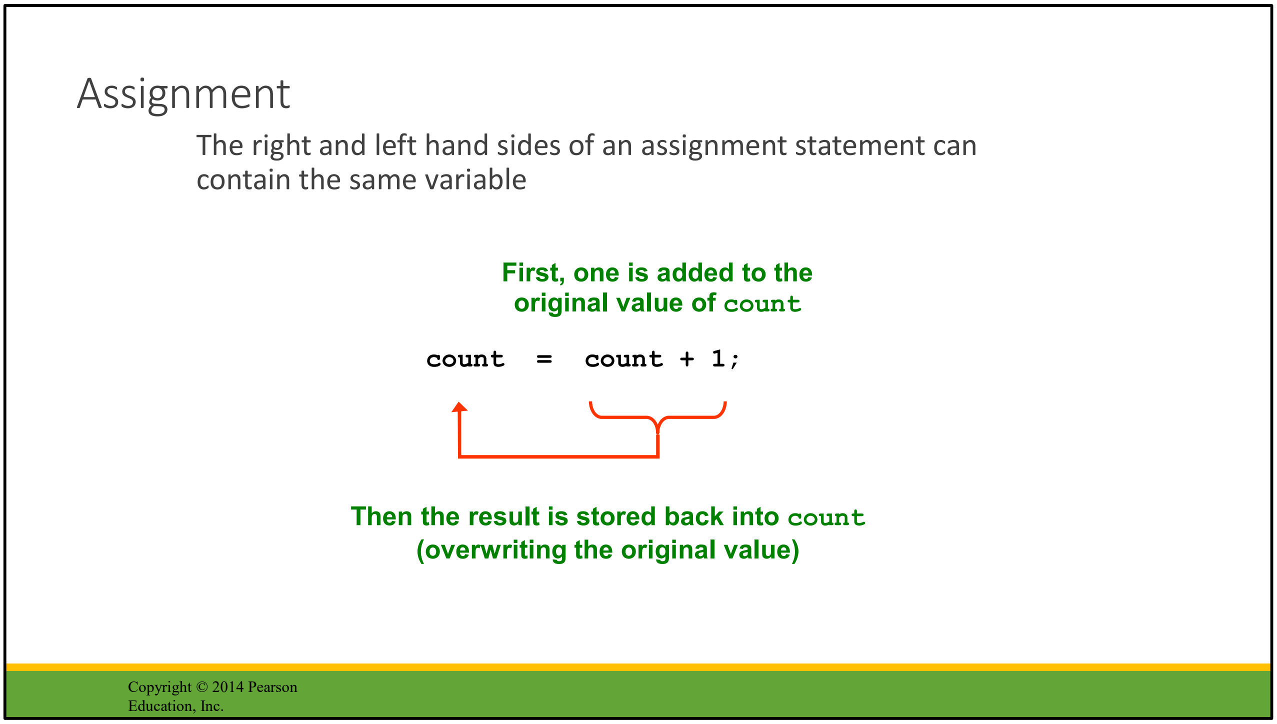
The assignment operator "=" has a lower precedence than the arithmetic operators

First the expression on the right hand side of the = operator is evaluated



```
answer  =  sum / 4 + MAX * lowest;
```

Then the result is stored in the variable on the left hand side



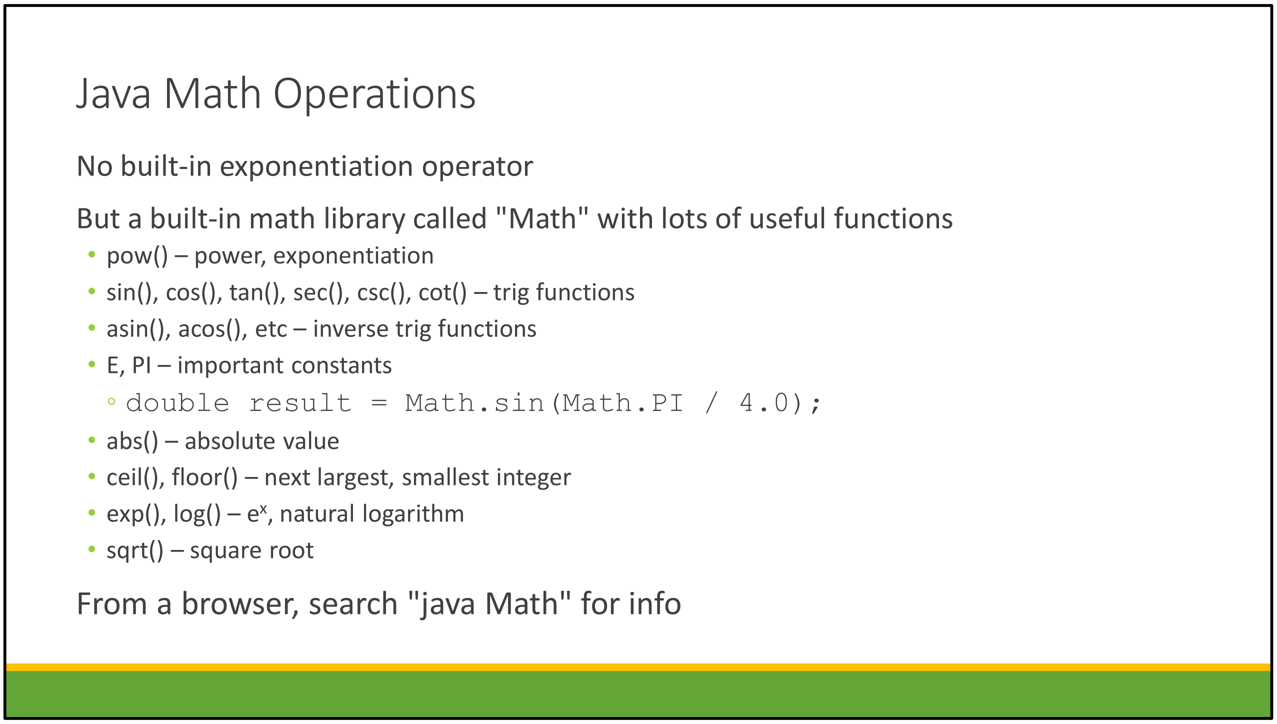
Assignment

The right and left hand sides of an assignment statement can contain the same variable

First, one is added to the original value of `count`

```
count  =  count + 1;
```

Then the result is stored back into `count` (overwriting the original value)



Java Math Operations

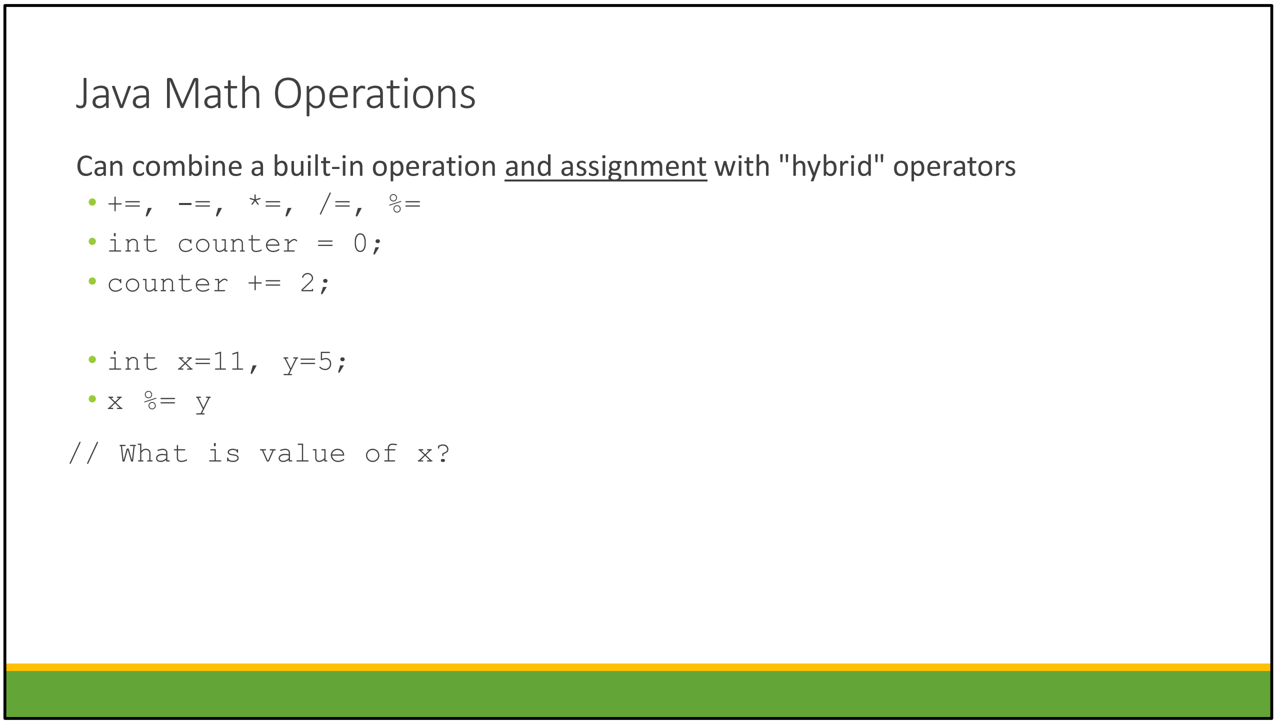
No built-in exponentiation operator

But a built-in math library called "Math" with lots of useful functions

- pow() – power, exponentiation
- sin(), cos(), tan(), sec(), csc(), cot() – trig functions
- asin(), acos(), etc – inverse trig functions
- E, PI – important constants
  - `double result = Math.sin(Math.PI / 4.0);`
- abs() – absolute value
- ceil(), floor() – next largest, smallest integer
- exp(), log() – e^x, natural logarithm
- sqrt() – square root

From a browser, search "java Math" for info

In general, search "java <<keyword>>"



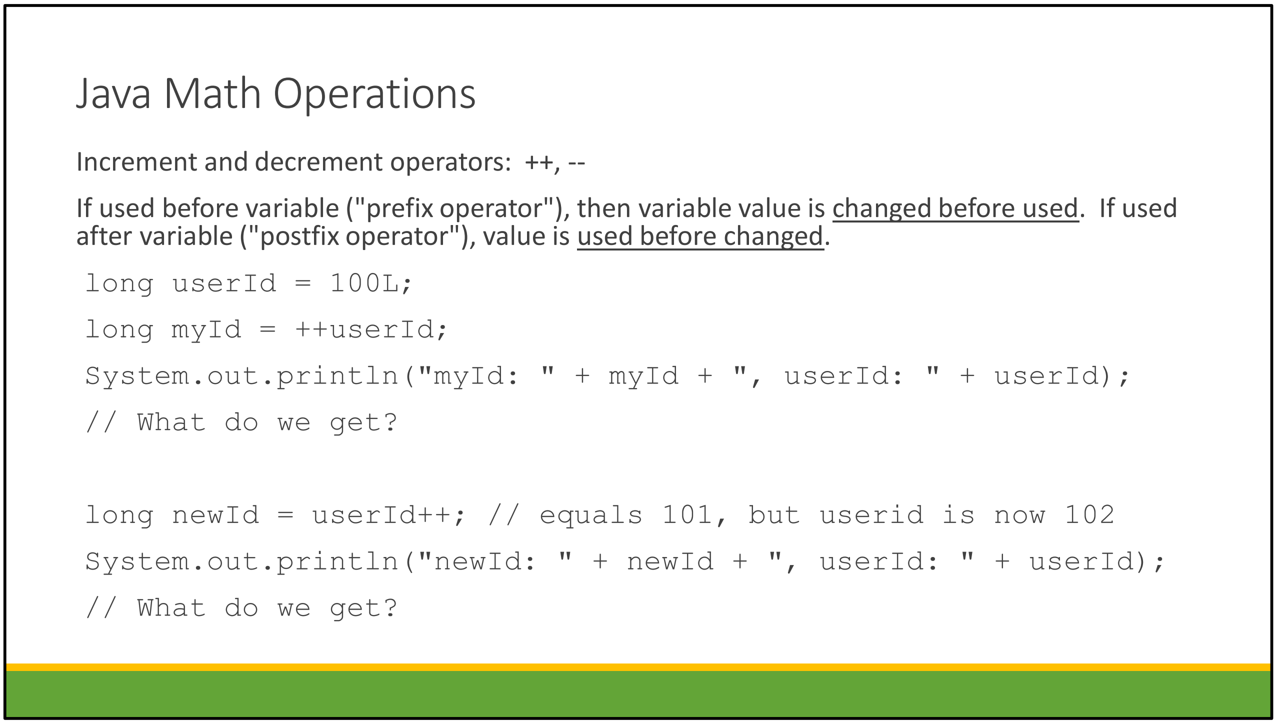
Java Math Operations

Can combine a built-in operation and assignment with "hybrid" operators

- `+=, -=, *=, /=, %=`
- `int counter = 0;`
- `counter += 2;`

- `int x=11, y=5;`
- `x %= y`

```
// What is value of x?
```



Java Math Operations

Increment and decrement operators: ++, --

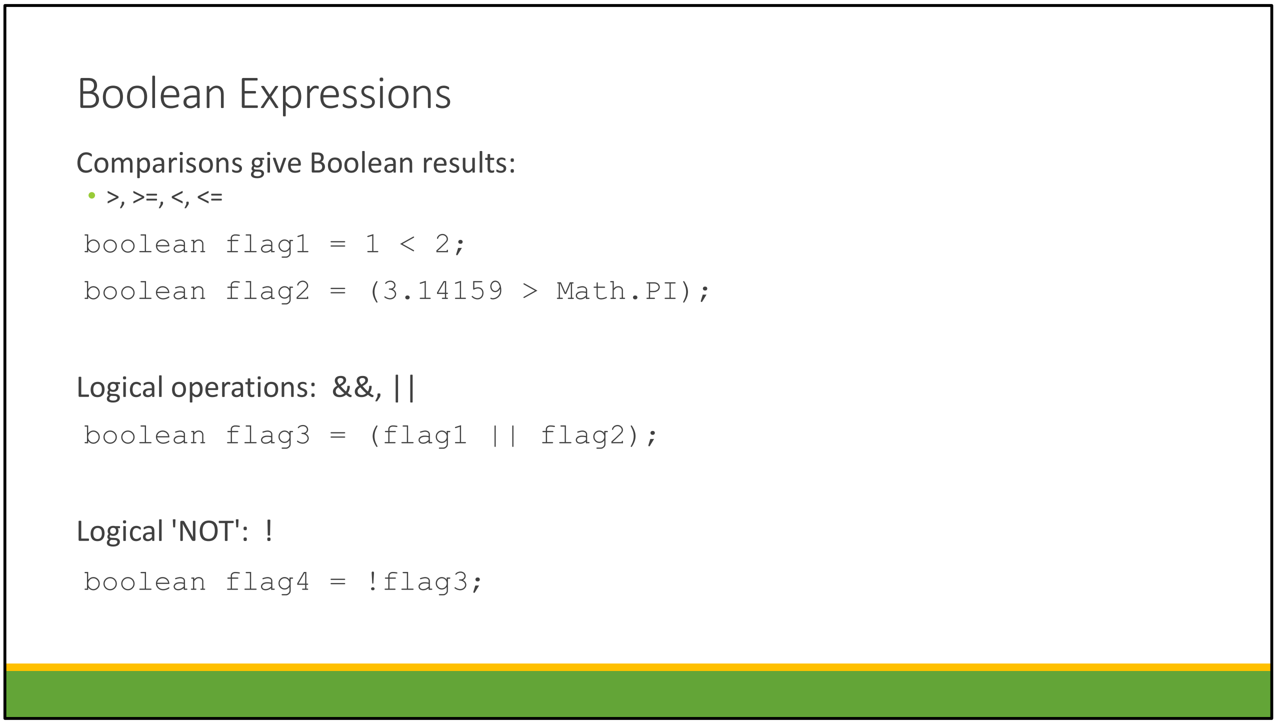
If used before variable ("prefix operator"), then variable value is changed before used. If used after variable ("postfix operator"), value is used before changed.

```
long userId = 100L;
long myId = ++userId;
System.out.println("myId: " + myId + ", userId: " + userId);
// What do we get?

long newId = userId++; // equals 101, but userid is now 102
System.out.println("newId: " + newId + ", userId: " + userId);
// What do we get?
```

Java order of operations:
- Prefix/Parentheses/Casting/Multiply,Divide/Add,Subtract/Assignment/Postfix

Boolean variables and Expressions



Boolean Expressions

Comparisons give Boolean results:

- >, >=, <, <=

```
boolean flag1 = 1 < 2;
boolean flag2 = (3.14159 > Math.PI);
```

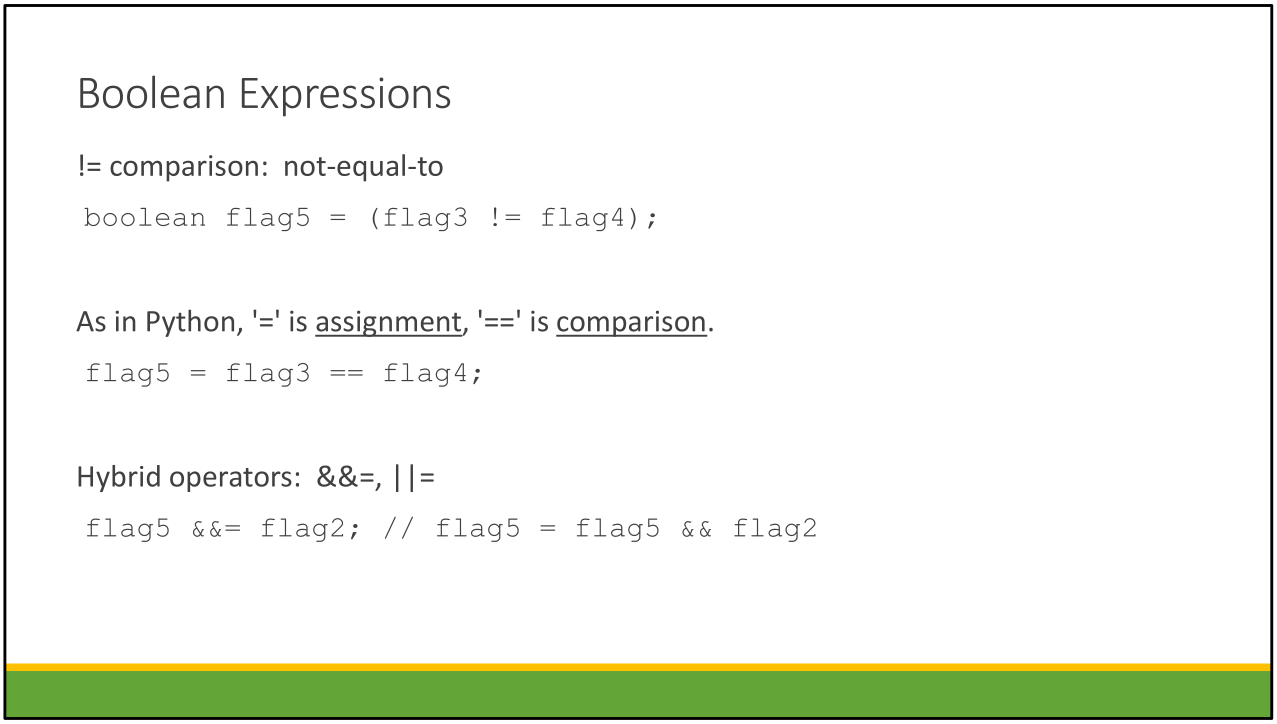
Logical operations: &&, ||

```
boolean flag3 = (flag1 || flag2);
```

Logical 'NOT': !

```
boolean flag4 = !flag3;
```

Review logical operations: and, or



Boolean Expressions

!= comparison: not-equal-to

```
boolean flag5 = (flag3 != flag4);
```

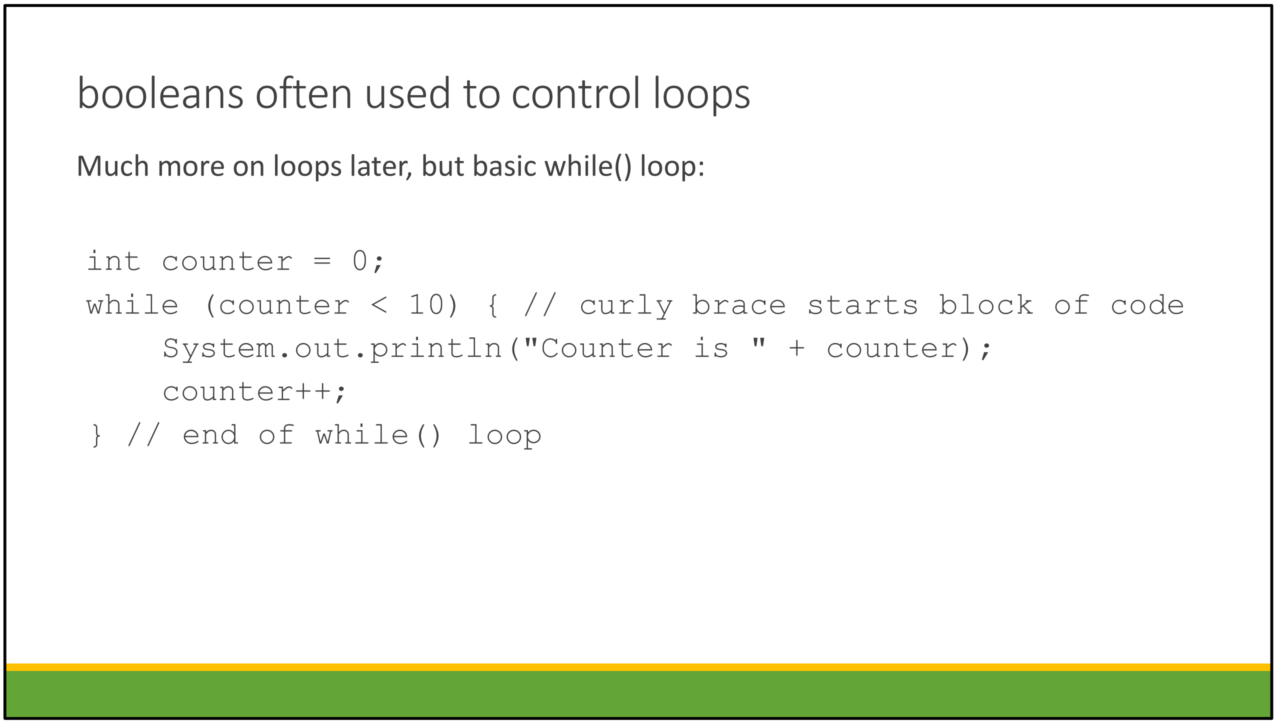
As in Python, '=' is assignment, '==' is comparison.

```
flag5 = flag3 == flag4;
```

Hybrid operators: &&=, ||=

```
flag5 &&= flag2; // flag5 = flag5 && flag2
```

SLOW WALKTHRU ON SECOND EXAMPLE!

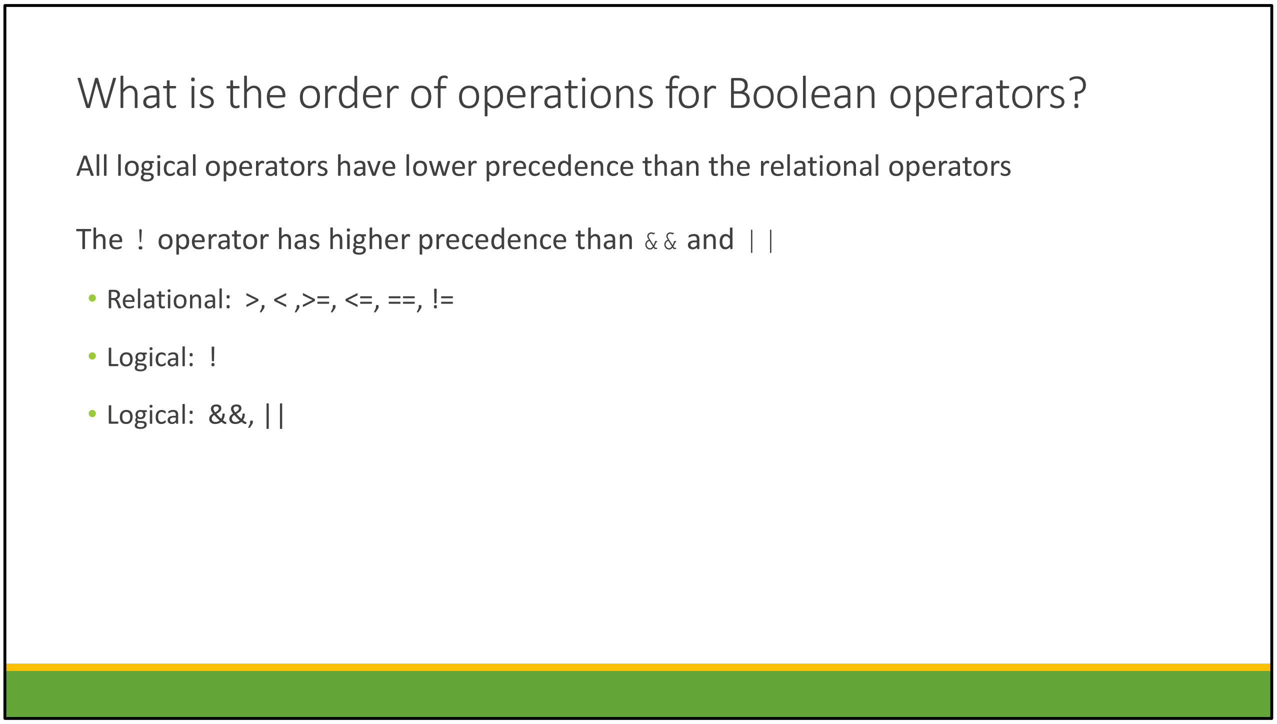


booleans often used to control loops

Much more on loops later, but basic while() loop:

```
int counter = 0;
while (counter < 10) { // curly brace starts block of code
    System.out.println("Counter is " + counter);
    counter++;
} // end of while() loop
```

FORMALLY, while() executes the following statement or block
ALWAYS use curly braces to define a code block. NEVER just 1 statement
BE CAREFUL to make sure your loop will terminate eventually

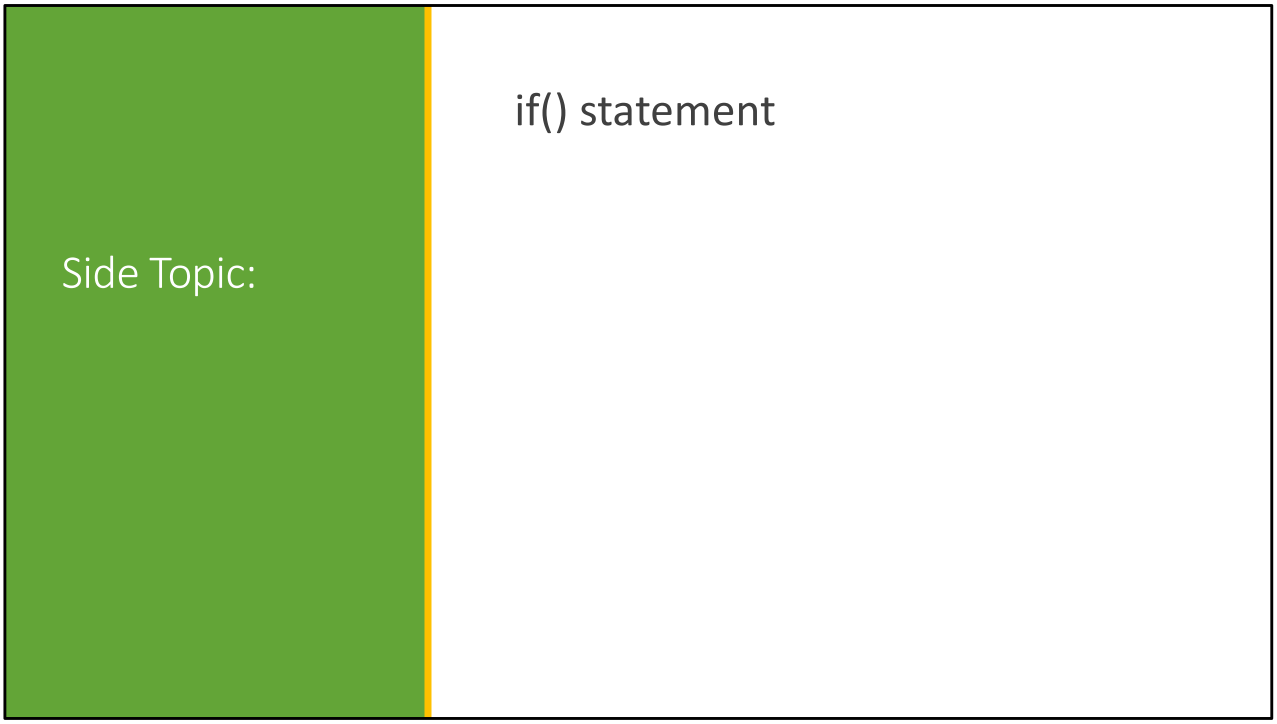


What is the order of operations for Boolean operators?

All logical operators have lower precedence than the relational operators

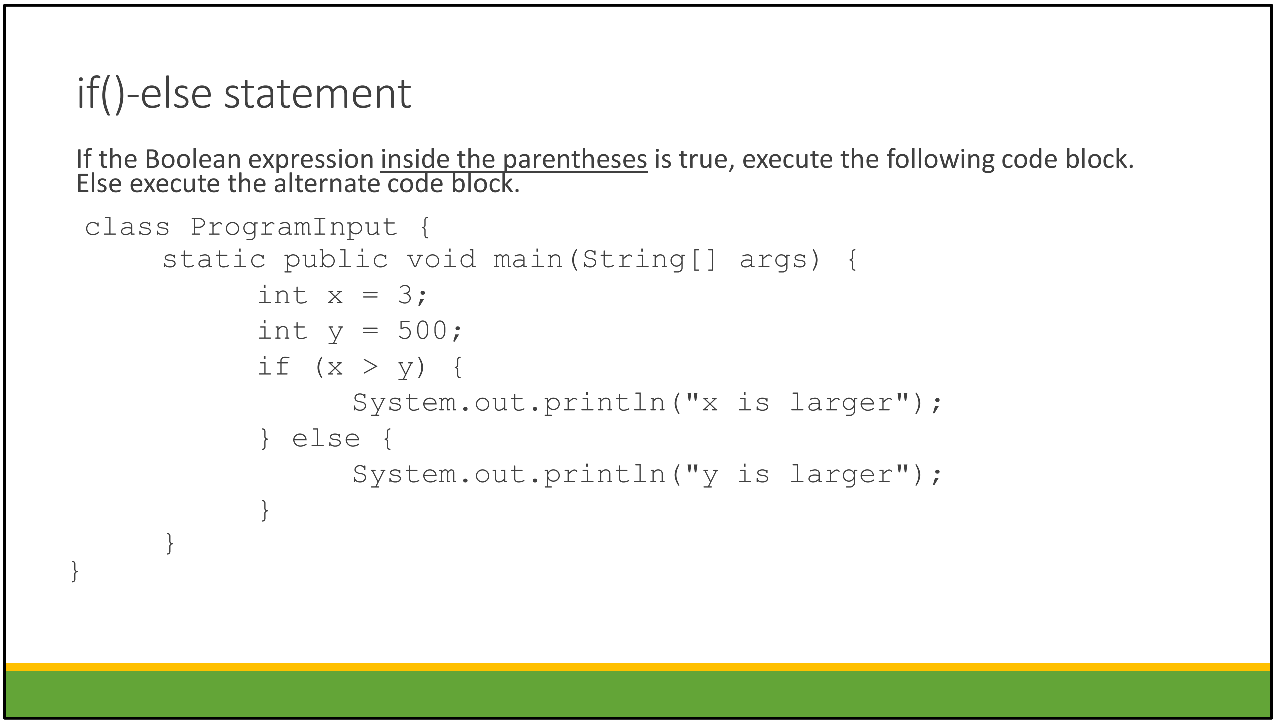
The `!` operator has higher precedence than `&&` and `||`

- Relational: >, < ,>=, <=, ==, !=
- Logical: !
- Logical: &&, ||



Side Topic:

if() statement



if()-else statement

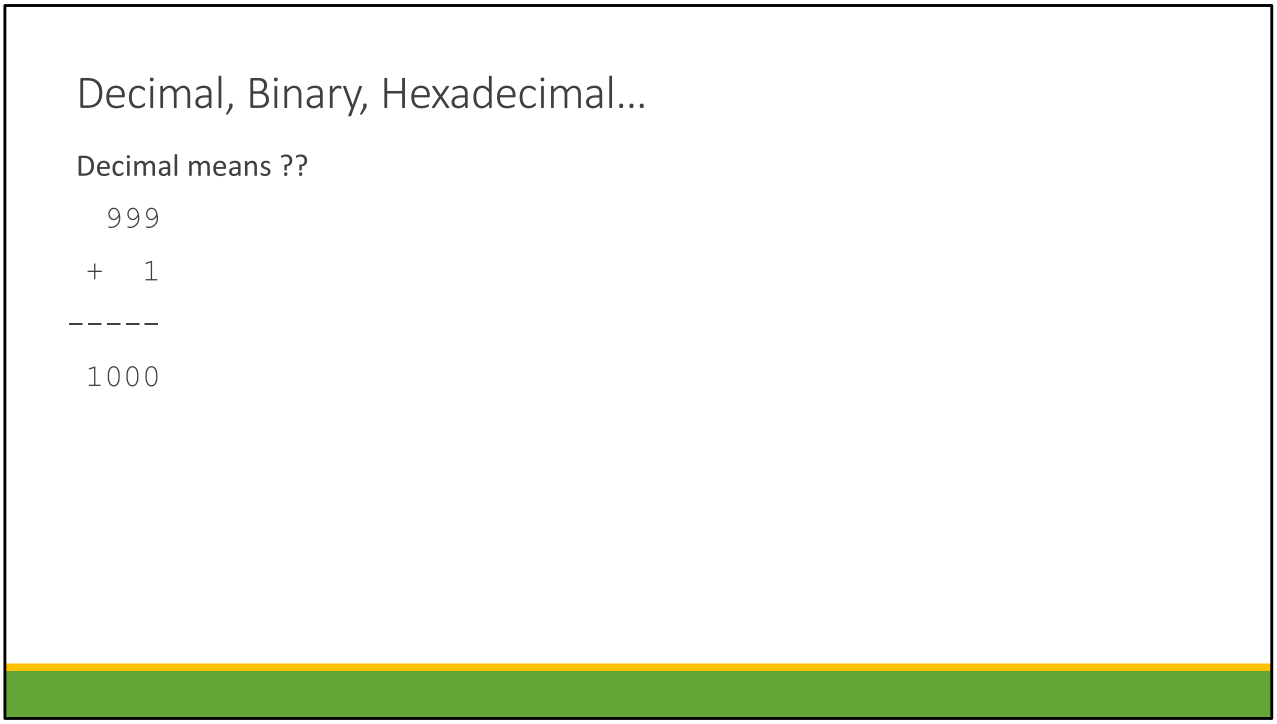
If the Boolean expression inside the parentheses is true, execute the following code block. Else execute the alternate code block.

```
class ProgramInput {
    static public void main(String[] args) {
        int x = 3;
        int y = 500;
        if (x > y) {
            System.out.println("x is larger");
        } else {
            System.out.println("y is larger");
        }
    }
}
```

ALTERNATIVE:
boolean yBigger = (y > x); if (yBigger) …

AS WITH while(), ALWAYS USE CURLY BRACES

Hexadecimal!

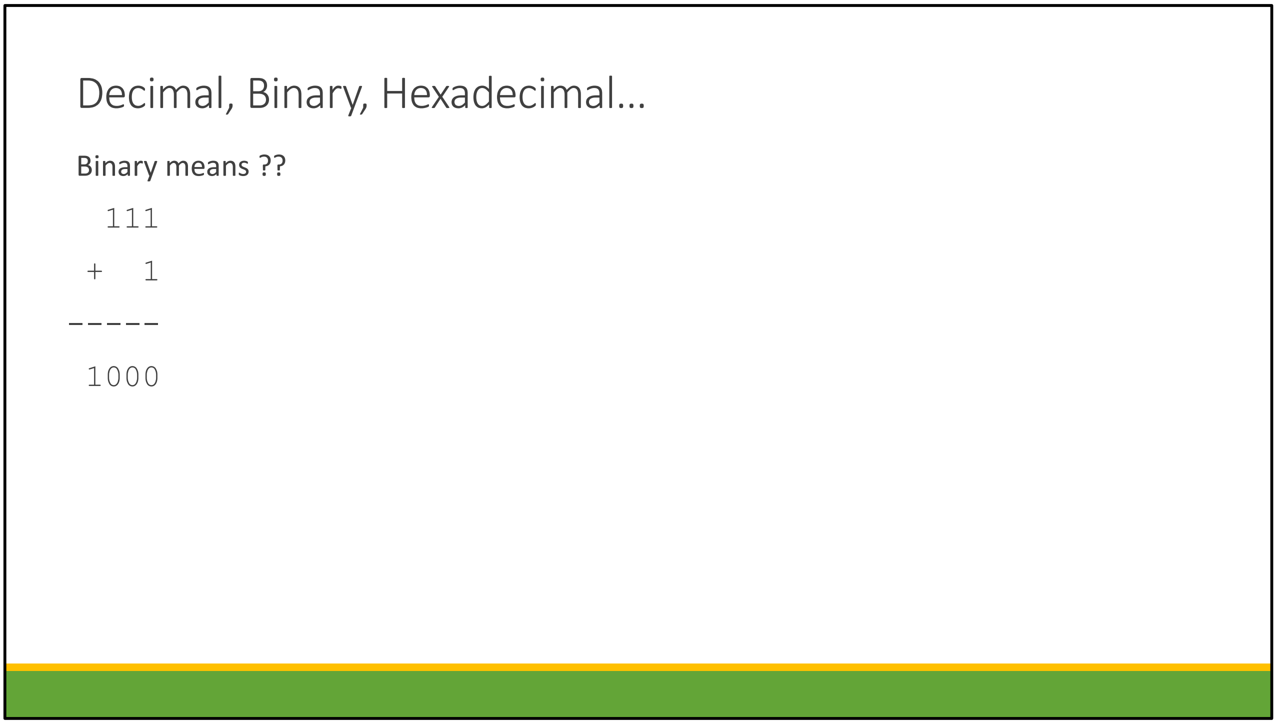


Decimal, Binary, Hexadecimal…

Decimal means ??

```
  999
+   1
-----
 1000
```

Ones place, tens place, hundreds place, …

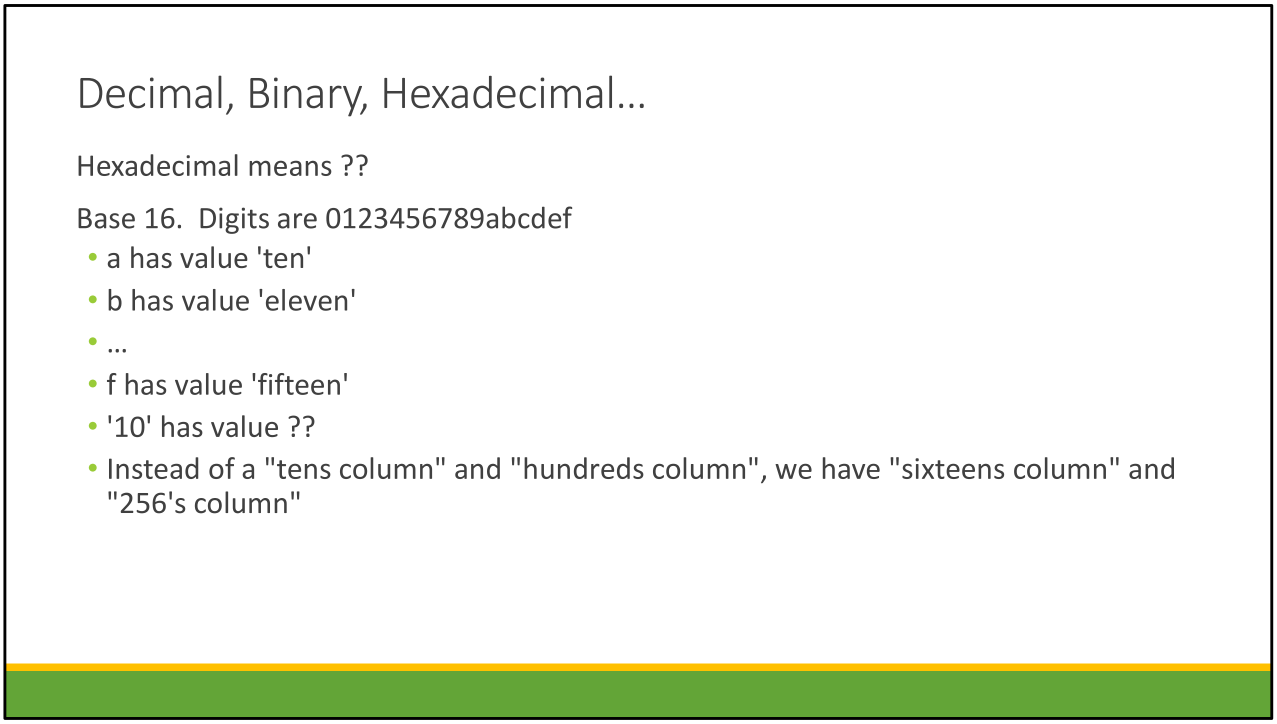


Decimal, Binary, Hexadecimal…

Binary means ??

```
  111
 +  1
-----
 1000
```

Ones place, twos place, fours place, eights, place, 16s place, …



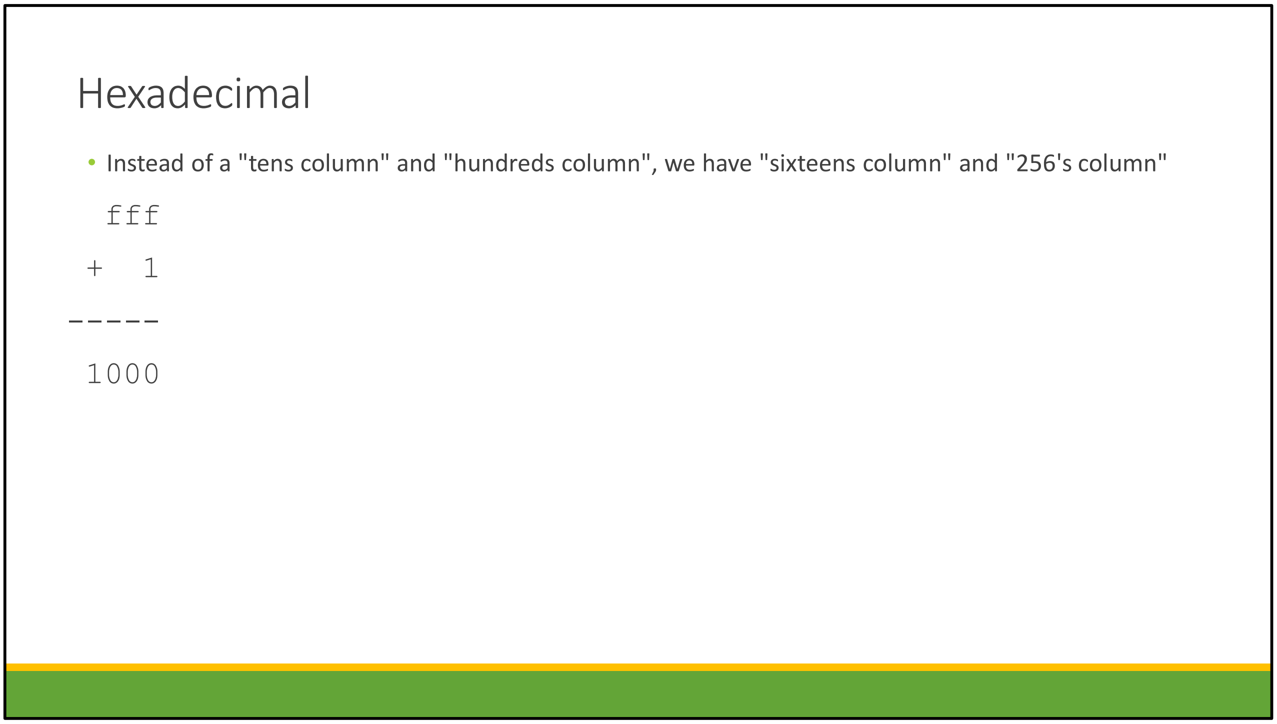
Decimal, Binary, Hexadecimal…

Hexadecimal means ??

Base 16. Digits are 0123456789abcdef

- a has value 'ten'
- b has value 'eleven'
- …
- f has value 'fifteen'
- '10' has value ??
- Instead of a "tens column" and "hundreds column", we have "sixteens column" and "256's column"

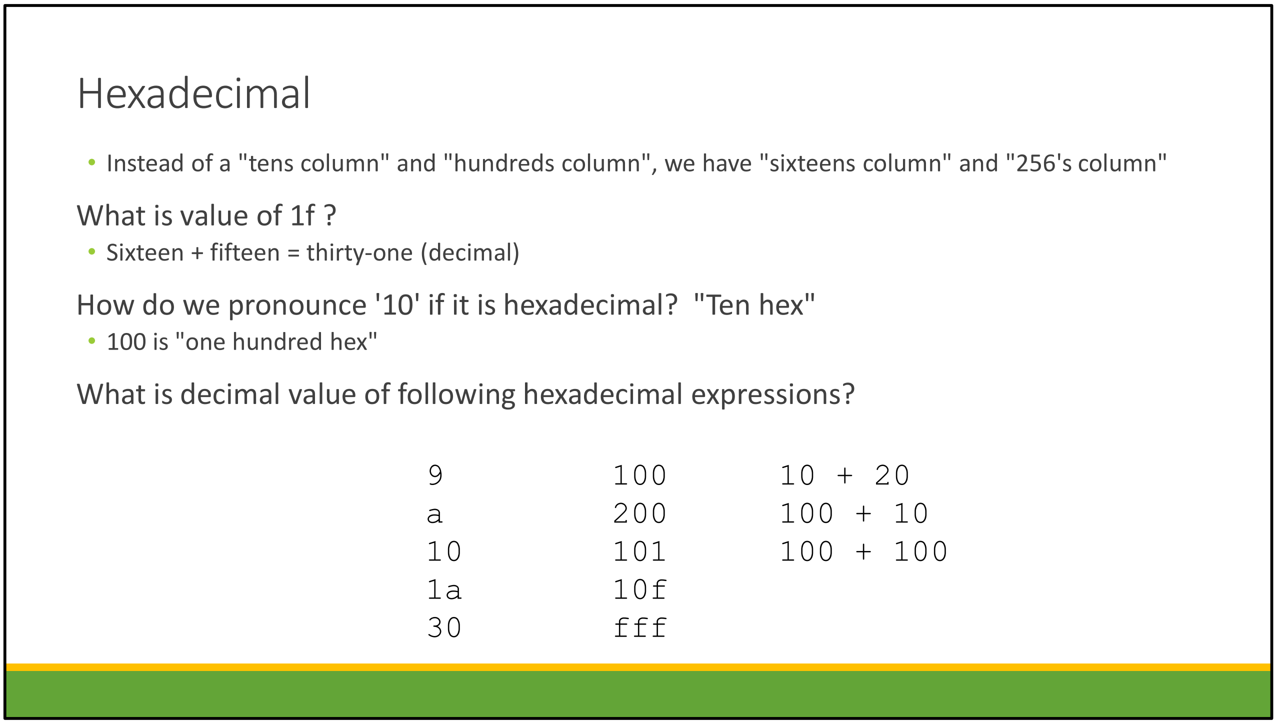
Write down some examples on board
Ones place, 16s place, 256s place, (16^3) place, …



Hexadecimal

- Instead of a "tens column" and "hundreds column", we have "sixteens column" and "256's column"

```
  fff
+   1
-----
 1000
```



Hexadecimal

- Instead of a "tens column" and "hundreds column", we have "sixteens column" and "256's column"

What is value of 1f ?

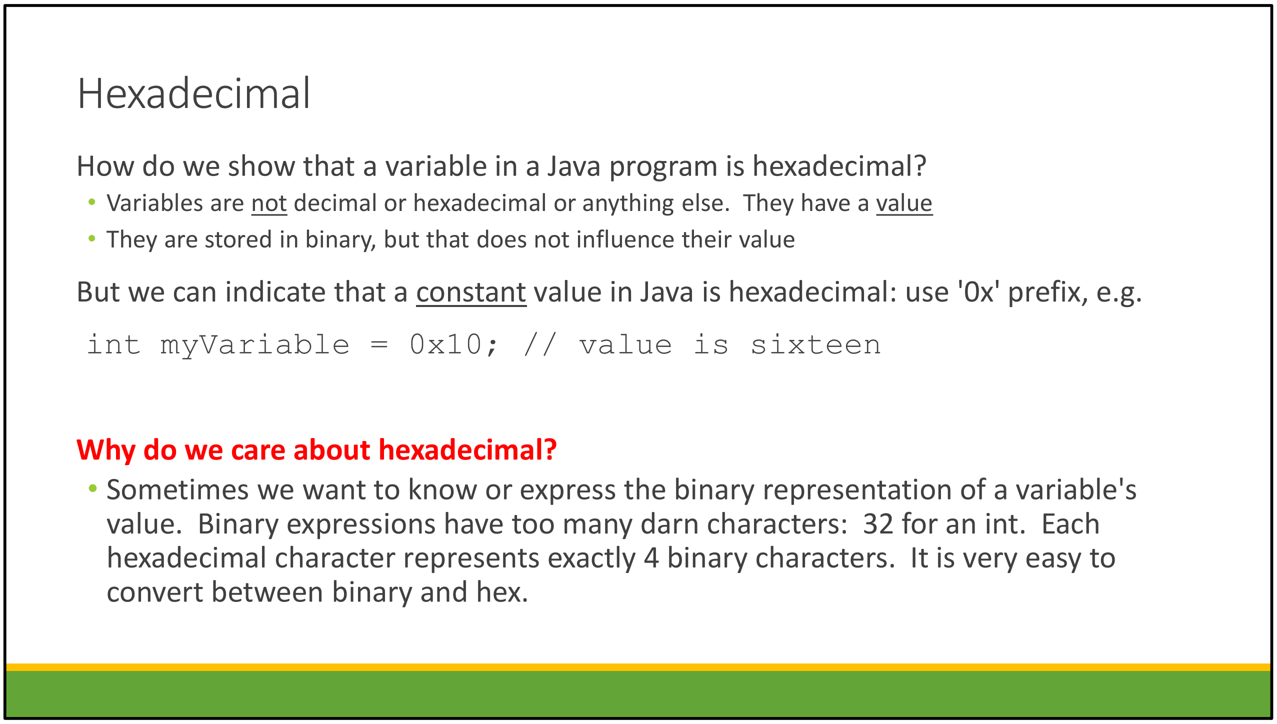
- Sixteen + fifteen = thirty-one (decimal)

How do we pronounce '10' if it is hexadecimal? "Ten hex"

- 100 is "one hundred hex"

What is decimal value of following hexadecimal expressions?

```
9          100        10 + 20
a          200        100 + 10
10         101        100 + 100
1a         10f
30         fff
```



Hexadecimal

How do we show that a variable in a Java program is hexadecimal?

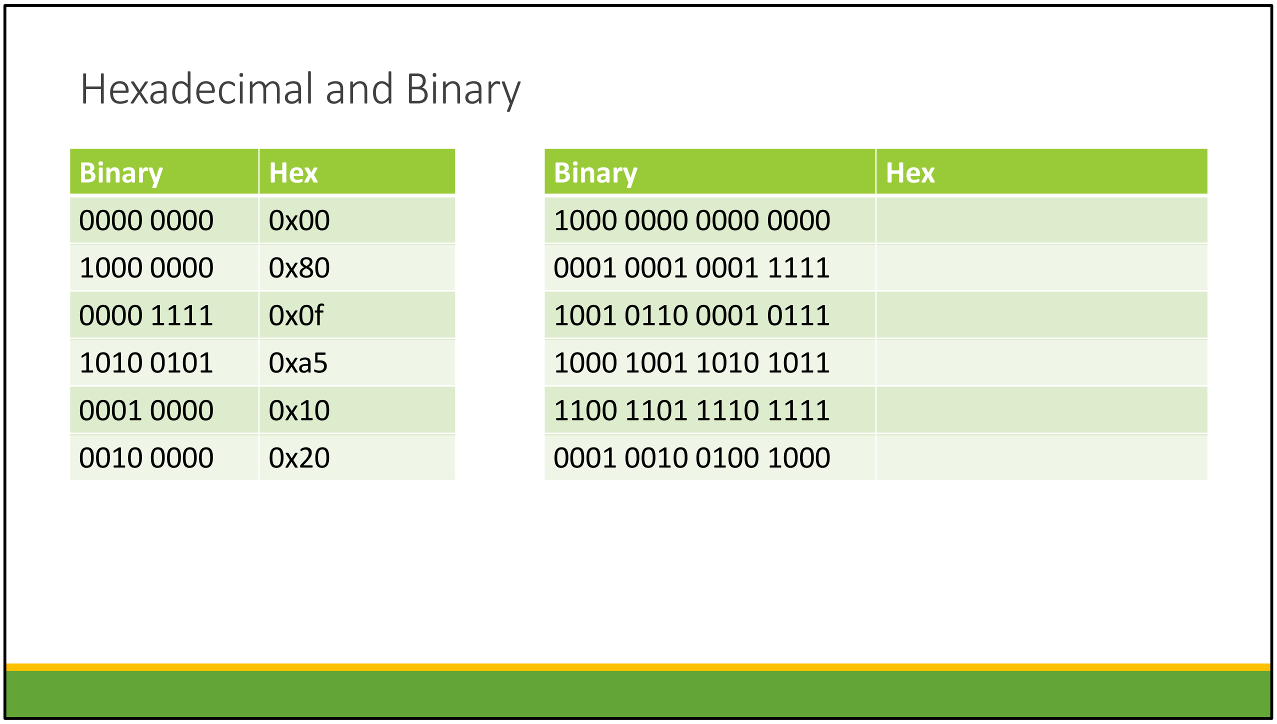
- Variables are not decimal or hexadecimal or anything else. They have a value
- They are stored in binary, but that does not influence their value

But we can indicate that a constant value in Java is hexadecimal: use '0x' prefix, e.g.

```
int myVariable = 0x10; // value is sixteen
```

Why do we care about hexadecimal?

- Sometimes we want to know or express the binary representation of a variable's value. Binary expressions have too many darn characters: 32 for an int. Each hexadecimal character represents exactly 4 binary characters. It is very easy to convert between binary and hex.



Hexadecimal and Binary

Binary	Hex
0000 0000	0x00
1000 0000	0x80
0000 1111	0x0f
1010 0101	0xa5
0001 0000	0x10
0010 0000	0x20

Binary	Hex
1000 0000 0000 0000	
0001 0001 0001 1111	
1001 0110 0001 0111	
1000 1001 1010 1011	
1100 1101 1110 1111	
0001 0010 0100 1000	

CALL ON STUDENTS TO ANSWER

Lab03

Now you write some programs…